MPI/C++

Point-to-Point Communication Routines

```
void MPI::Comm::Bsend(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
MPI::Prequest MPI::Comm::Bsend_init(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
void MPI::Attach_buffer(void* buffer, int size);
int MPI::Detach_buffer(void*& buffer);
void MPI::Request::Cancel(void) const;
int MPI::Status::Get_count(const MPI::Datatype& datatype) const;
int MPI::Status::Get_elements(const MPI::Datatype& datatype) const;
int MPI::Status::Get_source(void) const;
int MPI::Status::Get_tag(void) const;
int MPI::Status::Get_error(void) const;
MPI::Request MPI::Comm::Ibsend(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
bool MPI::Comm::Iprobe(int source, int tag) const;
MPI::Request MPI::Comm::Irecv(void *buf, int count, const MPI::Datatype& datatype, int source, int tag) const;
MPI::Request MPI::Comm::Irsend(const void *buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
MPI::Request MPI::Comm::Isend(const void *buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
MPI::Request MPI::Comm::Issend(const void *buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
void MPI::Comm::Probe(int source, int tag, MPI::Status& status) const;
void MPI::Comm::Recv(void* buf, int count, const MPI::Datatype& datatype, int source, int tag) const;
MPI::Prequest MPI::Comm::Recv_init(void* buf, int count, const MPI::Datatype& datatype, int source, int tag) const;
void MPI::Request::Free();
void MPI::Comm::Rsend(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
MPI::Prequest MPI::Comm::Rsend_init(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
void MPI::Comm::Send(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
MPI::Prequest MPI::Comm::Send_init(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
void MPI::Comm::Sendrecv(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype, int dest,
                         int sendtag, void* recvbuf, int recvcount, const MPI::Datatype& recvtype, int source, int recvtag) const;
void MPI::Comm::Sendrecv_replace(void* buf, int count, const MPI::Datatype& datatype, int dest, int sendtag, int source,
                                 int recvtag) const;
void MPI::Comm::Ssend(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
MPI::Prequest MPI::Comm::Ssend_init(const void* buf, int count, const MPI::Datatype& datatype, int dest, int tag) const;
void MPI::Prequest::Start();
void MPI::Prequest::Startall(int count, MPI::Prequest array_of_requests[]);
bool MPI::Request::Test(MPI::Status& status);
bool MPI::Status::Is_cancelled() const;
bool MPI::Request::Testall(int count, MPI::Request req_array[], MPI::Status stat_array[]);
bool MPI::Request::Testany(int count, MPI::Request array[], int& index, MPI::Status& status);
int MPI::Request::Testsome(int incount, MPI::Request req_array[], int array_of_indices[], MPI::Status stat_array[]);
void MPI::Request::Wait(MPI::Status& status);
void MPI::Request::Waitall(int count, MPI::Request req_array[], MPI::Status stat_array[]);
int MPI::Request::Waitany(int count, MPI::Request array[], MPI::Status& status);
int MPI::Request::Waitsome(int incount, MPI::Request req_array[], int array_of_indices[], MPI::Status stat_array[]);
```

Collective Communication Routines

```
void MPI::Comm::Allgather(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype, void* recvbuf,
                          int recvcount, const MPI::Datatype& recvtype) const;
void MPI::Comm::Allgatherv(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype, void* recvbuf,
                           const int recvcounts[], const int displs[], const MPI::Datatype& recvtype) const;
void MPI::Comm::Allreduce(const void* sendbuf, void* recvbuf, int count, const MPI::Datatype& datatype, const MPI::Op& op) const;
void MPI::Comm::Alltoall(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype, void* recvbuf,
                         int recvcount, const MPI::Datatype& recvtype) const;
void MPI::Comm::Alltoallv(const void* sendbuf, const int sendcounts[], const int sdispls[], const MPI::Datatype& sendtype,
                          void* recvbuf, const int recvcounts[], const int rdispls[], const MPI::Datatype& recvtype) const;
void MPI::Comm::Barrier() const;
void MPI::Comm::Bcast(void* buffer, int count, const MPI::Datatype& datatype, int root) const;
void MPI::Comm::Gather(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype, void* recvbuf,
                       int recvcount, const MPI::Datatype& recvtype, int root) const;
void MPI::Comm::Gatherv(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype, void* recvbuf,
                        const int recvcounts[], const int displs[], const MPI::Datatype& recvtype, int root) const;
void MPI::Op::Init(MPI::User_function *func, bool commute);
void MPI::Op::Free();
void MPI::Comm::Reduce(const void* sendbuf, void* recvbuf, int count, const MPI::Datatype& datatype, const MPI::Op& op,
                       int root) const;
void MPI::Comm::Reduce_scatter(const void* sendbuf, void* recvbuf, int recvcounts[], const MPI::Datatype& datatype,
                               const MPI::Op& op) const;
void MPI::Intracomm::Scan(const void *sendbuf, void *recvbuf, int count, const MPI::Datatype& datatype, const MPI::Op& op) const;
void MPI::Comm::Scatter(const void* sendbuf, int sendcount, const MPI::Datatype& sendtype,
                        void* recvbuf, int recvcount, const MPI::Datatype& recvtype, int root) const;
void MPI::Comm::Scatterv(const void* sendbuf, const int sendcounts[], const int displs[], const MPI::Datatype& sendtype,
                         void* recvbuf, int recvcount, const MPI::Datatype& recvtype, int root) const;
```

Process Group Routines

```
int MPI::Group::Get_size() const;
void MPI::Group::Translate_ranks(const MPI::Group& group1, int n, const int ranks1[], const MPI::Group& group2, int ranks2[]);
static MPI::Group MPI::Group::Union(const MPI::Group& group1, const MPI::Group& group2);
int MPI::Group::Get_rank() const;
MPI::Group MPI::Group::Range_incl(int n, const int ranges[][3]) const;
MPI::Group MPI::Group::Range_excl(int n, const int ranges[][3]) const;
void MPI::Group::Free();
MPI::Group MPI::Group::Incl(int n, const int ranks[]) const;
static MPI::Group MPI::Group::Intersect(const MPI::Group& group1, const MPI::Group& group2);
MPI::Group MPI::Group::Excl(int n, const int ranks[]) const;
static MPI::Group MPI::Group::Difference(const MPI::Group& group1, const MPI::Group& group2)
static int MPI::Group::Compare(const MPI::Group& group1, const MPI::Group& group2);
```

Communicators Routines

```
int MPI::Comm::Compare(const MPI::Comm& comm1, const MPI::Comm& comm2);
MPI::Intracomm MPI::Intracomm::Create(const MPI::Group& group) const;
MPI::Cartcomm MPI::Cartcomm::Dup() const;
MPI::Graphcomm MPI::Graphcomm::Dup() const;
MPI::Intercomm MPI::Intercomm::Dup() const;
MPI::Intracomm MPI::Intracomm::Dup() const;
void MPI::Comm::Free(void);
MPI::Group MPI::Comm::Get_group() const;
int MPI::Comm::Get_rank() const;
MPI::Group MPI::Intercomm::Get_remote_group() const;
int MPI::Intercomm::Get_remote_size() const;
int MPI::Comm::Get_size() const;
MPI::Intercomm MPI::Intercomm::Split(int color, int key) const;
MPI::Intracomm MPI::Intracomm::Split(int color, int key) const;
bool MPI::Comm::Is_inter() const;
MPI::Intercomm MPI::Intracomm::Create_intercomm(int local_leader, const MPI::Comm& peer_comm, int remote_leader, int tag) const;
MPI::Intracomm MPI::Intercomm::Merge(bool high);
```

Derived Types Routines

```
void MPI::Datatype::Commit();
MPI::Datatype MPI::Datatype::Create_contiguous(int count) const;
void MPI::Datatype::Get_extent(MPI::Aint& lb, MPI::Aint& extent) const;
void MPI::Datatype::Free();
MPI::Datatype MPI::Datatype::Create_hindexed(int count, const int array_of_blocklengths[],
                                             const MPI::Aint array_of_displacements[]) const
MPI::Datatype MPI::Datatype::Create_indexed(int count, const int array_of_blocklengths[], const int array_of_displacements[]) const;
MPI::Datatype MPI::Datatype::Create_hvector( int v1, int v2, Aint v3 ) const
int MPI::Datatype::Get_size() const;
static MPI::Datatype MPI::Datatype::Create_struct(int count, const int array_of_blocklengths[],
                                                  const MPI::Aint array_of_displacements[], const
MPI::Datatype MPI::Datatype::Create_vector(int count, int blocklength, int stride) const;

Primer: MPI::Datatype niz_tip = MPI::INT.Create_contiguous(5);
```

Basic Types

```
MPI::CHAR, MPI::SIGNED_CHAR, MPI::UNSIGNED_CHAR, MPI::BYTE, MPI::WCHAR, MPI::SHORT, MPI::UNSIGNED_SHORT, MPI::INT, MPI::UNSIGNED,
MPI::LONG, MPI::UNSIGNED_LONG, MPI::FLOAT, MPI::DOUBLE, MPI::LONG_DOUBLE, MPI::LONG_LONG_INT, MPI::UNSIGNED_LONG_LONG,
MPI::LONG_LONG, MPI::LONG_LONG_INT, MPI::PACKED, MPI::LB, MPI::UB
```

Operators

```
MPI::MAX, MPI::MIN, MPI::SUM, MPI::PROD, MPI::LAND, MPI::BAND, MPI::LOR, MPI::BOR, MPI::LXOR, MPI::BXOR, MPI::MINLOC, MPI::MAXLOC,
MPI::REPLACE, MPI::OP_NULL
```

Virtual Topology Routines

```
void MPI::Cartcomm::Get_coords(int rank, int maxdims, int coords[]) const;
MPI::Cartcomm MPI::Intracomm::Create_cart(int ndims, const int dims[], const bool periods[], bool reorder) const;
void MPI::Cartcomm::Get_topo(int maxdims, int dims[], bool periods[], int coords[]) const;
int MPI::Cartcomm::Map(int ndims, const int dims[], const bool periods[]) const;
int MPI::Cartcomm::Get_cart_rank(const int coords[]) const;
void MPI::Cartcomm::Shift(int direction, int disp, int &rank_source, int &rank_dest) const;
MPI::Cartcomm MPI::Cartcomm::Sub(const bool remain_dims[]) const;
int MPI::Cartcomm::Get_dim() const;
void MPI::Compute_dims(int nnodes, int ndims, int dims[]);
MPI::Graphcomm MPI::Intracomm::Create_graph(int nnodes, const int index[], const int edges[], bool reorder) const;
void MPI::Graphcomm::Get_topo(int maxindex, int maxedges, int index[], int edges[]) const;
int MPI::Graphcomm::Map(int nnodes, const int index[], const int edges[]) const;
void MPI::Graphcomm::Get_neighbors(int rank, int maxneighbors, int neighbors[]) const;
int MPI::Graphcomm::Get_neighbors_count(int rank) const;
void MPI::Graphcomm::Get_dims(int nnodes[], int nedges[]) const;
int MPI::Comm::Get_topology() const;
```

Environment Management Routines

```
void MPI::Comm::Abort(int errorcode);
MPI::Errhandler MPI::Comm::Get_errhandler( void ) constvoid MPI::Get_error_string(int errorcode, char* string, int& resultlen);
void MPI::Init(int& argc, char**& argv);
double MPI::Wtime();
MPI::Comm::Create_errhandler(MPI::Comm::ERRHANDLERFN* function)
void MPI::Finalize();
bool MPI::Is_initialized();
void MPI::Errhandler::Free();
int MPI::Get_error_class(int errorcode);
void MPI::Get_processor_name(char*& name, int& resultlen);
double MPI::Wtick();
```

Miscellaneous Routines

```
MPI::Aint MPI::Get_address(void* location)
int MPI::Comm::Create_keyval(MPI::Comm::COPYATTRFN* comm_cop y_attr_fn,
                             MPI::Comm::DELETEATTRFN* comm_delete_attr_fn, void* extra_state)
void MPI::Datatype::Pack(const void* inbuf, int incount, void* outbuf, int outsize, int& position, const MPI::Comm& comm) const;
int MPI::Datatype::Pack_size(int incount, const MPI::Comm& comm) const;
void MPI::Pcontrol(const int level, ...);
void MPI::Datatype::Unpack(const void* inbuf, int insize, void* outbuf, int outcount, int& position, const MPI::Comm& comm) const;
```